Rogalgol integer subset compiler.

1. General considerations.

This compiler is used as a tool to implement the full Rogalgol compiler. The integer subset compiler is written in its own language, so the restrictions mentioned hereafter apply to the text of the integer subset compiler also. The full and subset compiler use the same interpretive codes and nearly the same interpreter.

2. Restrictions to Algol-60.

1. No nested procedure declarations.
2. No deallocation of storage for arrays: arrays must be declared only global.
3. Arrays are one-dimensional only.
4. Block leaving jumps do not clean the stacks.
5. No own variables. o
6. No real variables.
7. No switches nor designatonial expressions. o
8. Procedure parameters must be simple Boolean or integer variables and may be called by value only. o
9. The controlled variable in a for statement must be insubscripted.
10. The only permissable form of a for statement is: for = k step e until m do.
11. No multiple assignments. o
12. Only 6 characters of an identifier are significant. o

Some of these restrictions (labelled with o) apply also to the full Rogalgol compiler.

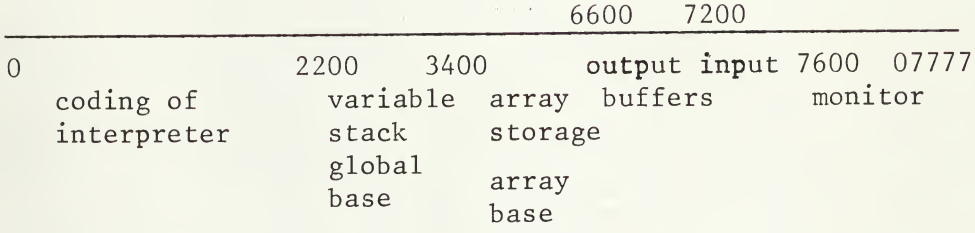
A further remark has to be made about actual parameters of procedures. The compiler must be able to determine the type of the actual parameter. When a procedure has a formal boolean parameter, the actual parameter might be a relational expression (e.g. 10> a). The compiler, however, first encounters the "10" which is an arithmetic expression and assumes that the further expression will be of type arithmetic also. Therefore it is necessary to inform the compiler beforehand about the type of the actual parameter: this can be done by prefixing the relational expression with false or.

is global to the procedure, it is addressed relative to the global base (= 20); if the variable is local to the procedure it is addressed relative to the local base (=22). As a few variables are reserved for saving status information (variable o = old local base, variable 1 = return address, variable 2 not used in the integer compiler but used in the full compiler for the blocklevel, variable 3 = procedure value) the maximum number of local variables in a procedure is 60.
During a procedure call the stack will show the following picture

	local N	free space pointer (= 23)
		local variables of procedure
	local 1	
direction	parameter N	
of		parameters of procedure
stackgrowth	parameter 1	
(ascending	proc. value	
address)	not used	
	return address	
	old value of 22	local base (= 22)
		amount of space required by main program (value of 21)
		global base (= 20)

The bottom of the array stack is initialized1200 words ahead of the bottom of the variable stack. When an array is declared, the space required for the elements of the array is taken off this stack by shifting up the array base.
The expression stack is initialized at the same address as the array stack, but this one grows downwards to lower addresses. Its stack pointer is in address 30.

The lay-out of field 0 is: (Field 1 is used entirely for interpretive code)



4. The interpretive codes.

The interpreter is able to interprete the following codes.

Non-implement codes stop the interpreter, print ↑ and wait for ↑P or ↑C.

Tos refers to the top of the expression stack where all arithmetic is done.

Octal	Decimal	Purpose + interface
0	0	No operation
1	1	declare array: on expr.stack: nr of decl., offset of array variable, lower + upperbound
2	2	select device: device nr: = tos
3	3	read : put signed integer from device on tos
4	4	store to local var, offset in next 6 bits (relative to 22)
5	5	print string: print following 6 bit codes as chars on dev, term. by 00
6	6	print : print tos as signed integer on dev.
7	7	chin : get next char dev, put it on tos
10	8	chout : put tos as char on dev.
11	9	jump : jump to location (12 bits) in next word
12	10	exit : leave procedure body
13	11	enter : enter procedure with no parameters
14	12	get value of local variable, offset in next 6 bits (rel to 22)
15	13	add
16	14	get value of arrayel: base add of array, index on expr.stack
17	15	store to arrayel
20	16	set tos to 12 bit constant
21	17	make tos: = - tos
22	18	unused

23	19	multiply	all arithmetic, boolean + rel. operators
24	20	divide	operate on top elements of expression
25	21	substract	stack and leave their result there
26	22	equal	
27	23	not equal	
30	24	less than	
31	25	greater than	
32	26	greater or equals	
33	27	less or equals	
34	28	jump if false: jump to loc in next 12 bits if tos =false	
35	29	set tos zero	
36	30	not: make tos: = not tos	
37	31	and	
40	32	or	
41	33	initialize interpreter: reset PC + local base 0 if global	
42	34	for statement: final values, increment, 1 if local contr. var., address of var.	
43	35	enter proc. with 1 param.	
44	36	enter proc. with tos = nr. of param.	
45	37	store tos in global variable, offset to global base in next 6 bits	
46	38	get value of global variable relative to 20	
47	39	disk:initialize in + output files	
50	40	skip : put CRLF to device	
51	41	skip next code	
52	42	get next 6 bits to tos as constant	
53	43	enter procedure with 1 parameters	
54	44	2	
55	45	3	
56	46	4	
57	47	not used, stop interpreter, print ↑	
77	63	and wait for ↑P, ↑C.	

The codes used internally in the compiler to represent the basic symbols are the same as in the full compiler.

6. Extensions to Algol-60.

The integer subset compiler accepts as alternatives to the < repeatable statement > also

while < boolean expression > do < statement >

and

do < statement > until < boolean expression >

this latter statement implies that comment after end is slipped up to ; end else until
no special codes are generated for these statements, so no extensions were necessary in the interpreter.

Disk (1) initializes the interpreter for disk input by fetching the sys device handler and look up of 3INALG.AL.
Disk (2) initializes for output by opening a tentative output file 3OUTAL.PA.
↑Z terminates the transfer when doing output: the output file is closed and made permanent. If no ↑Z is found, the contents of the output file are lost. Transfer on input is terminated when encountering EOF on the input file. Further reading produces ? 1725

7. Directions for use.

The integer subset compiler is generally present as a save file on the device: name is INTALG.SV
It can be run by typing
. R INTALG
the compiler identifies itself and asks for output device. If the answer is > 10, each time an identifier is encountered
its name, offset and type will be printed out on the tty.
As usual 1 means tty, 2 hsp and 3 system device.
The name of the output file will always be 3OUTAL.PA. Then the compiler asks for input device: 1 = tty, 2 hsr and 3 = system device. The input file name is always 3 INALG.AL. If this file cannot be found on the system device ?1634 will be issued. The compilation starts. When the message END OF COMPILATION is printed, the resulting output file must be assembled with PAL8 to produce a binary file. This binary file can be loaded by ABSLDR together with the binary code of the interpreter to yield a complete program. Possibly the EAE overlay is used also. In this way the compiler itself can also be compiled.

This can be represented schematically:

The interpreter and EAE overlay are normal PAL programs, which can be assembled using PAL8.

8. Error messages.

The compiler generates error messages in the form FAIL <error number > LINE < line number > IDENT < last used identifier >

Caution: the compiler is not very careful when issuing error messages, the real error can be in an other part of the program. Check especially if all worddelimeters are delimited by two '. When an error is encountered generation of code is stopped and the contents of the output file are lost.

Error number	Description	Issuer
1	identifier declared twice in the same block	sid
2	undeclared identifier	
3	missing after array identifier	subscript
4	missing after subscript list	subscript
5	more than 63 variables in procedure	
6	no $ after final end	
7	else expected in arithmetic expression	Ae
8	else expected in boolean expression	Be
9	Relational operator expected	Re
10	arithmetic primary cannot start with this	Aprime
11	integer divide does not have 2 integer operands	Aterm
12	) missing in arithmetic expression	Aprime
13	Controlled var. in for stmt is undeclared or subsc.	Statement
14	) missing in boolean expression	Bprint
15	Too many identifiers in scope at once	Sid
16	No end after block	Statement
17	Undeclared identifier in left part of assignment	Assignment

Error number	Description	Issuer
18	Error in declaration of array	Darr
19	Identifier does not start with letter	Ident
20	Not assigned error number	
21	Actual parameter list not closed by)	Pcall
22	do missing in for statement	Statement
23	no; after procedure body	Dproc
24	:= expected	Assignment
25	then expected	Ifclause
26	not assigned error number	
27	preliminary $ in input file	IN6
28	not assigned error number	
29	unspecified parameter	Pcall
30	Too many forward references	Sfr. Dlab
31	not assigned error number	
32	formal parameter list does not start with (	Pcall
33	do expected after while	Statement
34	until expected	Statement
40	error in specification of parameter	
43	no; after formal parameter specification	Dproc

the interpreter can also generate error messages.
These are fatal: the execution is stopped.

?0764	no "exit interpreter" allowed
?1313	illegal device on input
?1331	illegal device on output
?1616	error while fetch-ing system device handler
?1634	input file 3INALG.AL missing
?1650	no room available for output
?1746	error while reading input file
?2033	error while writing output file
?2124	error while closing output file
?1725	attempt to read beyond EOF on input file

9. Some instruction times.

Assuming 1.5 µsec memory cycle time, the following times were counted:

fetch of an interpretive code: av. 28.5 µsec

putting a result on the expression stack: 12.0 µsec

fetch of a 6-bit code (constant or offset): 27.0 µsec

fetch of a simple variable : 81.0 µsec

procedure call + return (no parameters): 148 µsec

jump : 43.5 µsec

fetch of an array element : 44.0 µsec